# Musterlösung zur Musterklausur Programmierung I

Daniel Appenmaier

26.11.2025

# Aufgabe 1: Theoriefragen

1. Bei der imperativen Programmierung steht die Frage "Wie soll das Ergebnis erreicht werden?"(1 Punkt) im Vordergrund, bei der deklarativen Programmierung die Frage "Was soll das Ergebnis sein?"(1 Punkt)

2. byte, short, int, long, float, double, char, boolean (je 0,5 Punkte)

3. Wird eine if-Verzweigung für eine Wertzuweisung verwendet, spricht man von einer bedingten Zuweisung (2 Punkte)

4. Felder sind spezielle Datenstruktur-Variablen, die zur Verarbeitung großer Mengen an Daten verwendet werden (1 Punkt). Die einzelnen Speicherplätze in einem Feld werden als Elemente bezeichnet, die über einen Index angesprochen werden können (1 Punkt).

| 0 | 1 | 2 | 3 | 4 | 5 | *Index* |
|---|---|---|---|---|---|---|
| A | X | U | A | T | L | *Element* |

Tabelle 1: Indexzeile (1 Punkt), Elementzeile (1 Punkt)

5. Eine Kategorie von ähnlichen Objekten bezeichnet man als Klasse (1 Punkt). Konkrete Ausprägungen bzw. Instanzen einer Klasse werden als Objekte bezeichnet (1 Punkt). Die Eigenschaften von Objekten werden als Attribute bezeichnet (1 Punkt). Das Verhalten von Objekten wird als Methoden bezeichnet (1 Punkt).

6. Das Diamatenproblem zeigt die Probleme der Mehrfachverberung auf (1 Punkt). Erbt eine Klasse über mehrere mögliche Pfade von einer Basisklasse und werden dabei möglicherweise Methoden der Basisklasse unterschiedlich überschrieben, entstehen dadurch nicht eindeutige Varianten also Mehrdeutigkeiten (1 Punkt).

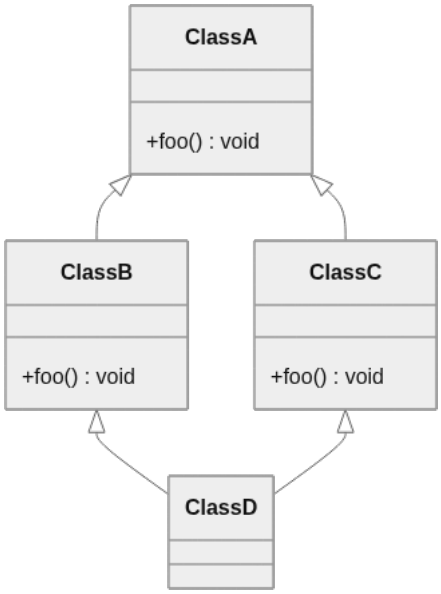


Abbildung 1: ClassA - ClassD (je 0,5 Punkte)

## Aufgabe 2: Prozeduraler Programmentwurf

**Klasse *ExamTask02***

```
public class ExamTask02 { // 0,5

  private static Dice[] dices; // 0,5
  private static int winningPoints; // 0,5
  private static int currentPoints; // 0,5
  private static int rounds; // 0,5
  private static Scanner scanner; // 0,5

  public static void main(String[] args) { // 0,5
    scanner = new Scanner(System.in); // 1
    dices = new Dice[5]; // 0,5

    for (int i = 0; i < dices.length; i++) { // 1
      dices[i] = new Dice(); // 1
    }

    System.out.print("Bitte Gewinnpunkte eingeben: "); // 0,5
    winningPoints = scanner.nextInt(); // 0,5

    while (currentPoints < winningPoints) { // 1
      rounds++; // 0,5
      System.out.print("Bitte Farbe (Rot, Gruen oder Blau) eingeben: "); // 0,5
      String color = scanner.next(); // 0,5

      int red = 0; // 0,5
      int green = 0; // 0,5
      int blue = 0; // 0,5

      for (int i = 0; i < dices.length; i++) { // 1
        dices[i].rollTheDice(); // 0,5
        switch (dices[i].getColor()) { // 1
          case RED -> red += dices[i].getValue(); // 1,5
          case GREEN -> green += dices[i].getValue(); // 1,5
          case BLUE -> blue += dices[i].getValue(); // 1,5
        }
      }

      System.out.println("Wurfwerte Rot: " + red); // 0,5
      System.out.println("Wurfwerte Gruen: " + green); // 0,5
      System.out.println("Wurfwerte Blau: " + blue); // 0,5

      if (red >= green && red >= blue && color.equals("Rot")) { // 1
        currentPoints++; // 0,5
        System.out.println("Richtig, Du bekommst einen Punkt"); // 0,5
      } else if (green >= red && green >= blue && color.equals("Gruen")) { // 1
        currentPoints++; // 0,5
        System.out.println("Richtig, Du bekommst einen Punkt"); // 0,5
      } else if (blue >= red && blue >= green && color.equals("Blau")) { // 1
        currentPoints++; // 0,5
        System.out.println("Richtig, Du bekommst einen Punkt"); // 0,5
      } else { // 0,5
        System.out.println("Leider falsch, Du bekommst leider keinen Punkt"); // 0,5
      }
    }

    System.out.println("Du hast " + rounds + " Runden benoetigt"); // 0,5
  } // 25

} // 28
```

# Aufgabe 3: Objektorientierter Progammentwurf

**Klasse *WeatherStation***

```
public class WeatherStation { // 0,5

  private static int counter; // 0,5
  private final int id; // 0,5
  private final String location; // 0,5
  private final List<MeasurementValue> measurementValues; // 0,5

  public WeatherStation(String location) { // 0,5
    counter++; // 0,5
    id = counter; // 0,5
    this.location = location; // 0,5
    measurementValues = new ArrayList<>(); // 0,5
  } // 2,5

  public void addMeasurementValue(MeasurementValue measurementValue) { // 0,5
    measurementValues.add(measurementValue); // 1
  } // 1,5

  public double getAverageValueBySensor(Sensor sensor, long start, long end) { // 0,5
    int total = 0; // 0,5
    int counter = 0; // 0,5
    for (MeasurementValue v : measurementValues) { // 1
      if (v.getSensor().equals(sensor)) { // 1
        if (v.getTimestamp() >= start && v.getTimestamp() <= end) { // 1,5
          total += v.getValue(); // 1
          counter++; // 0,5
        }
      }
    }
    return (double) total / counter; // 1
  } // 7,5

  public int getId() { // 0,5
    return id; // 0,5
  } // 1

  public String getLocation() { // 0,5
    return location; // 0,5
  } // 1

  public List<MeasurementValue> getMeasurementValues() { // 0,5
    return measurementValues; // 0,5
  } // 1

  @Override // +0,25
  public String toString() { // 0,5
    return id + ", " + location + ": " + measurementValues; // 1
  } // 1,5

} // 18,5
```

**Klasse *ValueHelper***

```
public class ValueHelper { // 0,5

  public static List<MeasurementValue> getValues(File file) // 0,5
      throws FileNotFoundException { // +0,5
    List<MeasurementValue> values = new ArrayList<>(); // 0,5

    Scanner scanner = new Scanner(file); // 0,5

    while (scanner.hasNextLine()) { // 1
      String line = scanner.nextLine(); // 1
      String[] tokens = line.split(";"); // 1

      long timestamp = Long.valueOf(tokens[0]); // 1
      double value = Double.valueOf(tokens[1]); // 1
      Sensor sensor = Sensor.valueOf(tokens[2]); // 1

      MeasurementValue value2 = new MeasurementValue(timestamp, sensor, value); // 1
      values.add(value2); // 0,5
    }

    scanner.close(); // 0,5

    return values; // 0,5
  } // 10

} // 10,5
```

# Aufgabe 4: Objektorientierter Programmentwurf 2

**Klasse *Present***

```
public abstract class Present implements Wrappable { // 1,5

  private boolean isWrapped; // 0,5

  public boolean isWrapped() { // 0,5
    return isWrapped; // 0,5
  } // 1

  @Override // +0,25
  public void unwrap() { // 0,5
    isWrapped = false; // 0,5
  } // 1

  @Override // +0,25
  public void wrap() { // 0,5
    isWrapped = true; // 0,5
  } // 1

} // 2
// 5
```

**Klasse *Book***

```
public class Book extends Present { // 1

  private final String title; // 0,5
  private final String author; // 0,5

  public Book(String title, String author) { // 0,5
    this.title = title; // 0,5
    this.author = author; // 0,5
  } // 1,5

  @Override // +0,25
  public String getDescription() { // 0,5
    return title + " " + author; // 1
  } // 1,5

} // 2
// 5
```

**Klasse *SantaClaus***

```
public class SantaClaus { // 0,5

  private final List<Present> presents; // 0,5

  public SantaClaus() { // 0,5
    this.presents = new ArrayList<>(); // 0,5
  } // 1

  public void addPresent(Present present) { // 0,5
    presents.add(present); // 0,5
  } // 1

  public int getNumberOfToys() { // 0,5
    int total = 0; // 0,5
    for (Present p : presents) { // 1
      if (p instanceof Toy) { // 1
        total++; // 0,5
      }
    }
    return total; // 0,5
  } // 4

  public void wrapAllPresents() { // 0,5
    for (Present p : presents) { // 1
      p.wrap(); // 0,5
    }
  } // 2

} // 1
// 9
```

**Klasse *ExamTask04***

```
public class ExamTask04 { // 0,5

  public static void main(String[] args) { // 0,5

    SantaClaus santa = new SantaClaus(); // 0,5

    santa.addPresent(new Toy("Elektrische Eisenbahn", 199)); // 1
    santa.addPresent(new Book("Es", "Stephen King")); // 1

    santa.wrapAllPresents(); // 0,5

    System.out.println("Anzahl Spielzeuge: " + santa.getNumberOfToys()); // 1

  } // 4,5

} // 0,5
// 5
```